ISEN
OUEST

PROJET SYSTEME COMPLEXE S6

OXYMETRIE DE POULS

-PARTIE INFORMATIQUE-

Version 9.2 mise à jour le vendredi 14 février 2025

TABLE DES MATIERES

1 INTRODUCTION

Le projet A3 2024-2025 consiste à réaliser un équipement médical appelé « oxymétre de pouls » qui permet de mesurer la fréquence cardiaque et la saturation en oxygène du sang (SpO2) d'un individu grâce à un capteur photo-électrique non invasif situé en général à l'extrémité du doigt du patient [1].

Comme le montre la Figure 1, la partie informatique consiste à développer un module permettant de calculer et d'afficher la fréquence cardiaque et le taux de Sp02 à partir des mesures d'absorption de la lumière rouge et de l'infrarouge fournies par le module « interface capteur oxymètre ».

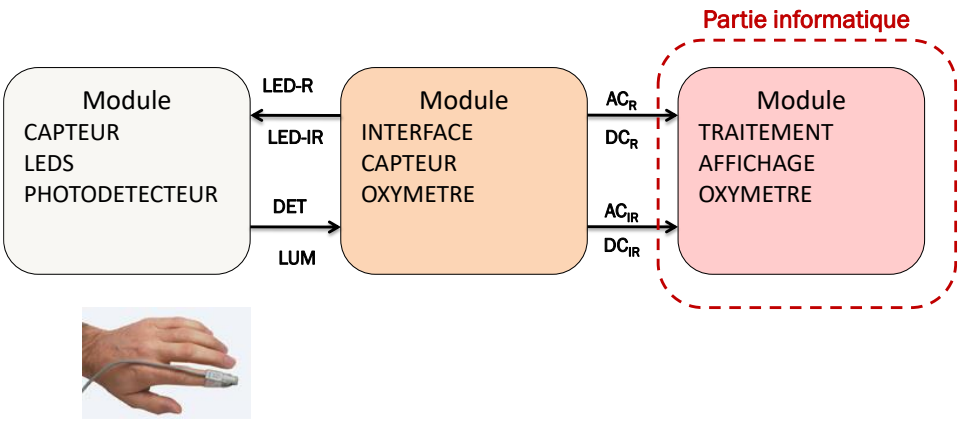


FIGURE 1 : SCHEMA GENERAL DU PROJET N3

Les mesures d'absorption fournies pour le module précédent sont les suivantes :

- AC_R : variation crête à crête de la lumière rouge mesurée (onde de pouls)
- DC_R : valeur moyenne de la lumière rouge mesurée (tissus)
- AC_{IR} : variation crête à crête de la lumière infrarouge mesurée (onde de pouls)
- DC_{IR} : valeur moyenne de la lumière infrarouge mesurée (tissus)

Ces 4 mesures sont transmises tous les 2 ms par liaison USB au module informatique (le format de trame sera décrit plus loin)

Le module informatique, intitulé « Traitement et affichage oxymètre » doit effectuer les opérations suivantes (voir Figure 2) :

- Lecture sur périphérique USB des signaux AC_R , DC_R, AC_{IR} et DC_{IR}
- Filtrage numérique des signaux
- Mesure du Sp02 et de la fréquence cardiaque

- Affichage des mesures via une interface Python fournie par l'équipe enseignante.

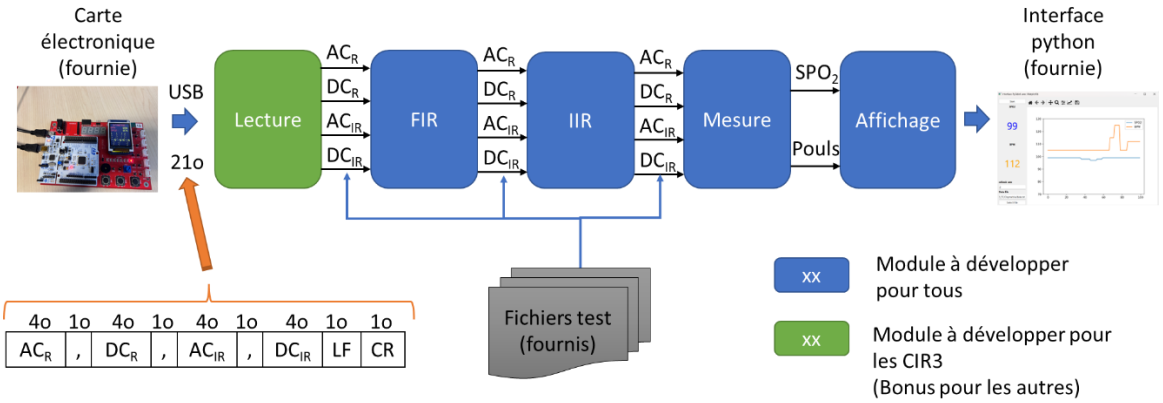


FIGURE 2 : SYNOPTIQUE DE LA PARTIE INFORMATIQUE

Afin de pouvoir fonctionner en temps réel, on demande que votre programme soit capable de calculer un résultat pour chaque mesure d'oxymétrie donnée en entrée du programme. Autrement dit pour une mesure (AC_R, DC_R, AC_{IR} et DC_{IR}) votre programme doit sortir et afficher un résultat de SP02 et de rythme cardiaque (voir Figure 3). Il va de soi que les premiers résultats seront erronés du fait du manque de mesures prises en compte.

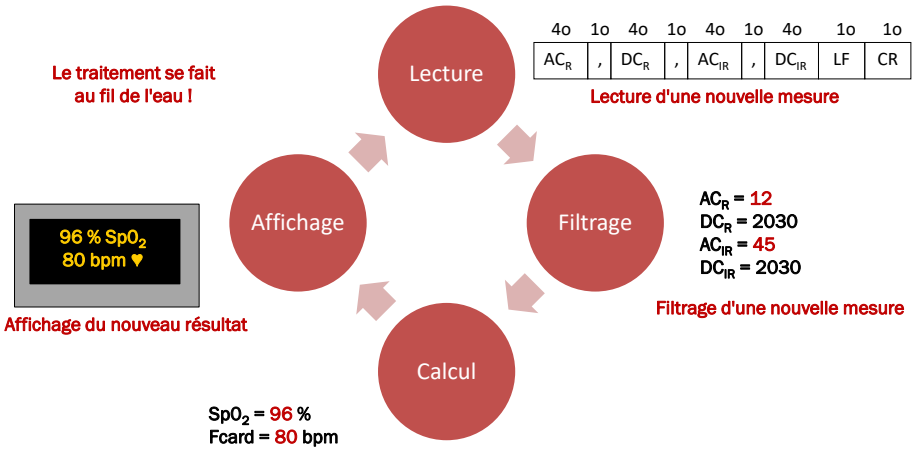


FIGURE 3 : ENCHAINEMENT DES BLOCS

2 DESCRIPTION DES BLOCS

2.1 LECTURE DES CHAMPS

Cette partie est à faire pour les CIR3 et optionnelle pour les autres cycles.

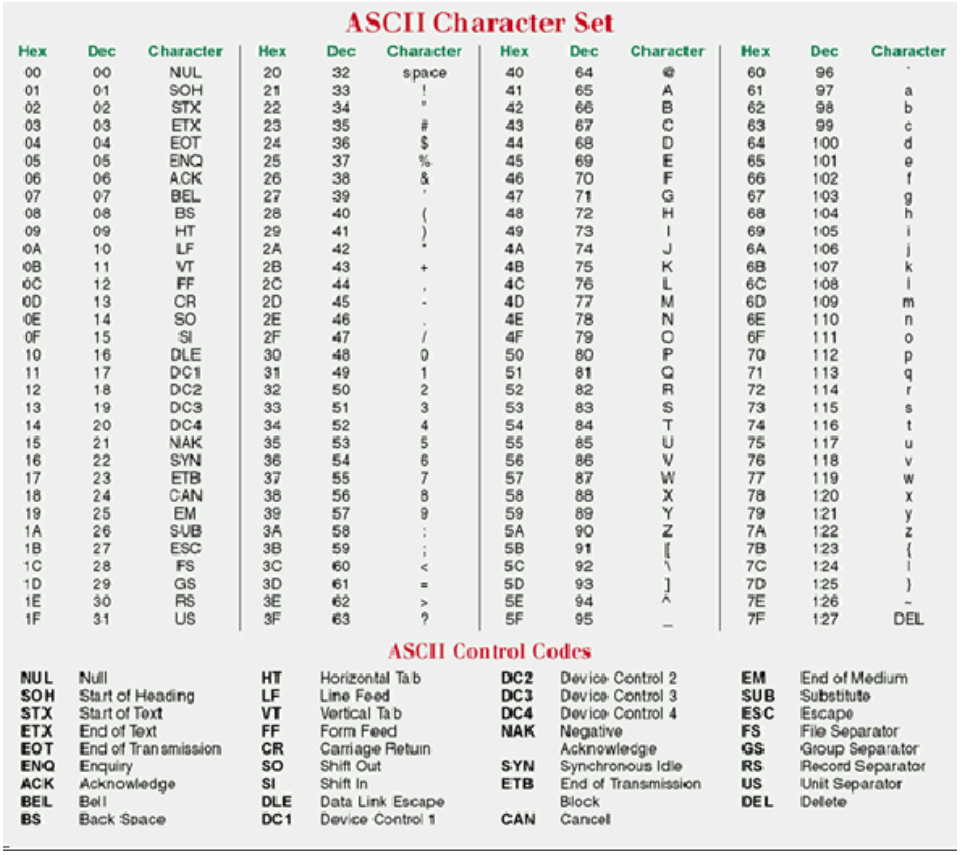
2.1.1 Codage d'une trame de mesure

Les mesures en provenance du module « interface capteur oxymètre » sont stockées sous le format suivant :

Taille (octets)	4	1	4	1	4	1	4	1	1
Champ	AC_R	,	DC_R	,	AC_{IR}	,	DC_{iR}	LF	CR

TABLEAU 1 : FORMAT DE LECTURE DES CHAMPS (21 OCTETS PAR LIGNE)

La carte électronique écrit toutes les 2 ms une ligne de mesure sur le port USB de la carte. Chaque champ est écrit sous format texte, i.e. sous la forme d'une série de caractères. Les valeurs numériques (AC_R , DC_R, AC_{IR} et DC_{IR}) quant à elles sont écrites chacune sous la forme de 4 caractères représentant des valeurs allant de « 0000 » à « 4095 ». On rappelle que tout caractère est représenté par son code ASCII dont on donne la table en Figure 4. On note ainsi que les champs « , », « LF » et « CR » sont des caractères auxquels sont associés des codes ASCII.



ASCII Character Set

Hex	Dec	Character	Hex	Dec	Character	Hex	Dec	Character	Hex	Dec	Character
00	00	NUL	20	32	space	40	64	@	60	96	`
01	01	SOH	21	33	!	41	65	A	61	97	a
02	02	STX	22	34	"	42	66	B	62	98	b
03	03	ETX	23	35	#	43	67	C	63	99	c
04	04	EOT	24	36	$	44	68	D	64	100	d
05	05	ENQ	25	37	%	45	69	E	65	101	e
06	06	ACK	26	38	&	46	70	F	66	102	f
07	07	BEL	27	39	'	47	71	G	67	103	g
08	08	BS	28	40	(	48	72	H	68	104	h
09	09	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	DEL

ASCII Control Codes

NUL	Null	HT	Horizontal Tab	DC2	Device Control 2	EM	End of Medium
SOH	Start of Heading	LF	Line Feed	DC3	Device Control 3	SUB	Substitute
STX	Start of Text	VT	Vertical Tab	DC4	Device Control 4	ESC	Escape
ETX	End of Text	FF	Form Feed	NAK	Negative Acknowledge	FS	File Separator
EOT	End of Transmission	CR	Carriage Return	SYN	Synchronous Idle	GS	Group Separator
ENQ	Enquiry	SO	Shift Out	ETB	End of Transmission Block	RS	Record Separator
ACK	Acknowledge	SI	Shift In	CAN	Cancel	US	Unit Separator
BEL	Bell	DLE	Data Link Escape			DEL	Delete
BS	Back Space	DC1	Device Control 1				

FIGURE 4 : TABLE DU CODE ASCII

Le Tableau 2 montre un exemple d'une ligne mesure.

Champ	**ACR**				**,**	**DCR**				**,**	**ACIR**				**,**	**DCIR**				**LF**	**CR**
Valeur	2	0	8	5	,	2	0	3	0	,	2	0	2	7	,	2	0	3	0	LF	CR
Code ASCI (hexa)	32	30	38	35	2C	32	30	33	30	2C	32	30	32	37	2C	32	30	33	30	0A	0D

TABLEAU 2 : EXEMPLE D'UNE LIGNE DE MESURE

Point sensible : Etablir la synchronisation afin de récupérer les valeurs.

2.1.2 Relevé USB

2.1.2.1 INSTALLATION DE L'ENVIRONNEMENT DE PROGRAMMATION WINDOWS.

Idée : avant de changer d'environnement de DEV, faites une sauvegarde complète de votre projet fonctionnel, ceci pour éviter d'avoir un projet qui ne fonctionne plus du tout sous Windows…

Si vous utilisez un IDE de programmation (VS CODE, CLION) vérifiez au préalable si votre installation comprend un compilateur compatible Windows de type `MinGW`. Si c'est le cas, dans les settings de votre projet, changez votre compilateur `gcc` de WSL pour celui compatible Windows.

Sinon, installez le compilateur `MinGW`. Un document explicatif d'installation de `MinGW` est disponible sur Moodle. Sinon référez-vous aux explications fournies sur le web.

Avant de passer à l'étape suivante, vérifiez que votre projet fonctionne bien sous Windows avec `MinGW`.

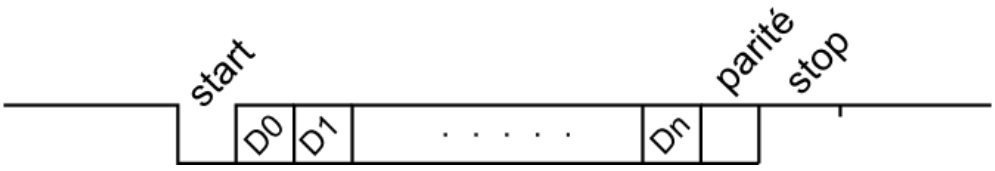
2.1.2.2 DEVELOPPEMENT DE LA COMMUNICATION AVEC LA LIB FTD2XX.

On trouve de la documentation pour écrire le code ici

https://ftdichip.com/software-examples/code-examples/c-builder/

L'envoi de chaque trame se fait en UART, c'est un protocole simple dit « série » qui était autrefois le protocole qu'on utilisais pour envoyer des informations sur un unique fil électrique (le protocole est hérité du télégraphe). Aujourd'hui les ordinateurs envoient plusieurs informations en parallèle grâce à des prises et câbles possédant de multiples fils (par exemple en USB4 il y a trois canaux de données en parallèle, chacun dédoublés car étant une paire différentielle), mais la maitrise fine de cela dépasse le cadre de notre cours. Heureusement, pour ceux désirant simplement transmettre des informations en série simple (comme en USB1, en USB2 ou sur un connecteur série RS-232) il suffit de télécharger ce pilote ftd2xx qui nous permet d'utiliser l'UART sur n'importe quelle connexion USB.

Une trame UART ressemble à ceci :



Les bits D0 à D1 correspondent à des bouts de la trame que vous avez construite précédemment, le nombre de bit est à définir quand on construit la liaison. Mais en plus de votre trame, ils sont eux même entourés d'identifications supplémentaires, notamment un bit de start qui définit le début du signal et un bit de stop qui en définit la fin.

Il y a également un bit de parité optionnel, qui est comme une mini somme de contrôle. C'est la somme des bits précédent, sur un unique bit, qui définit donc uniquement si la somme est paire ou impaire. C'est un système de contrôle assez rudimentaire.

Il est aussi important de noter que le bit de stop peut durer un certain temps (1 temps de bit normal, ou 1.5 ou même 2.).

Notez également que du fait de l'héritage des transmissions télégraphique, le niveau de base de la ligne est 1 et non 0. Le bit de start est donc un 0 qui vient couper la ligne, et non un 1.

Enfin, tout ceci est transmis à un certain rythme, un nombre de bit par seconde. On parle ici plus généralement de « symbole par seconde » et l'unité de cela est le baud.

Tout ceci définit notre façon de communiquer et il faut évidemment que ce soit en accord avec ce que la carte STM32 attends, sinon ça ne fonctionnera pas ou mal.

Notre carte STM32 fonctionne à 9600 Baud, 8 bits de données, un bit de stop d'une longueur de 1 et pas de contrôle de la parité (vu qu'on a déjà notre somme de contrôle). On doit également préciser qu'elle n'utilise pas de « flow control », un système de contrôle permettant d'éviter à un récepteur de recevoir trop de donnée (ce qui n'arrivera pas ici).

Vous allez devoir vous référer à la documentation ftd2xx pour comprendre comment paramétrer la connexion afin de respecter ce que la carte STM32 attend.

Les drivers FTDI ainsi que la procédure d'installation (sous Windows) sont disponible sur l'intranet.

2.1.3 Conversion

Chaque champ est représenté par une valeur entière comprise entre 0 et 4095. La Figure 5 et la Figure 6 montrent un exemple de mesures obtenues avec les composantes AC_R et DC_R.

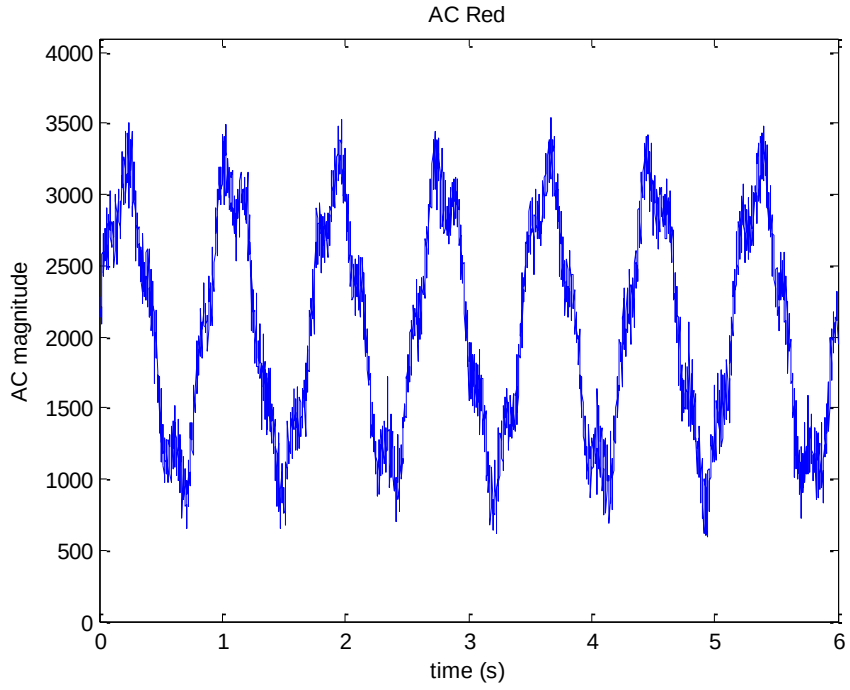


FIGURE 5 : EXEMPLE DE MESURE OBTENUE POUR LA COMPOSANTE AC_R

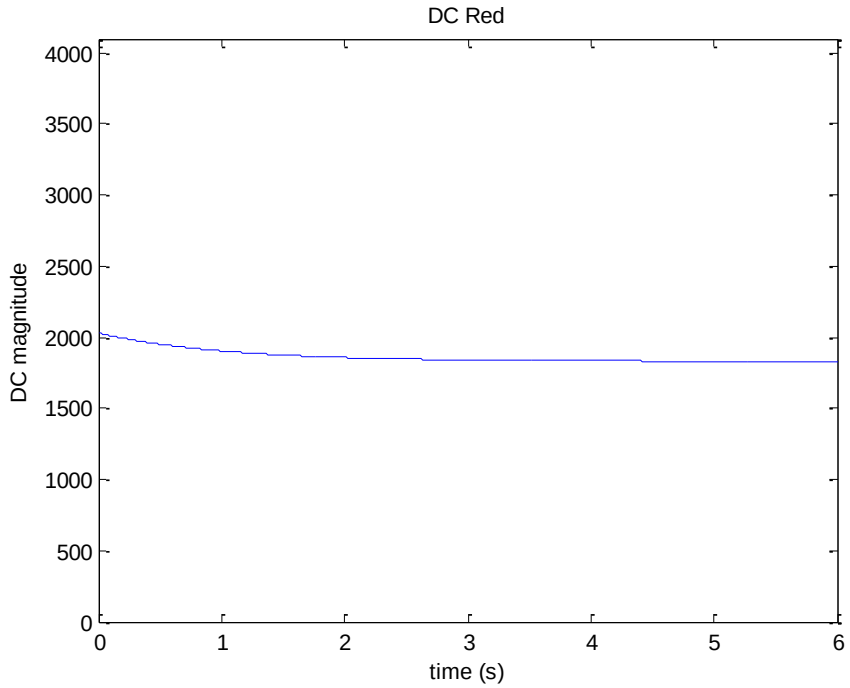


FIGURE 6 : EXEMPLE DE MESURE OBTENUE POUR LA COMPOSANTE DC_R

Afin de rendre possible les opérations du filtrage il convient de recadrer les 2 composantes alternatives (AC_R et AC_{IR}) autour de 0, comme le montre la Figure

7. Ce recadrage peut être approximatif, il sert simplement à faciliter les équations des filtres.

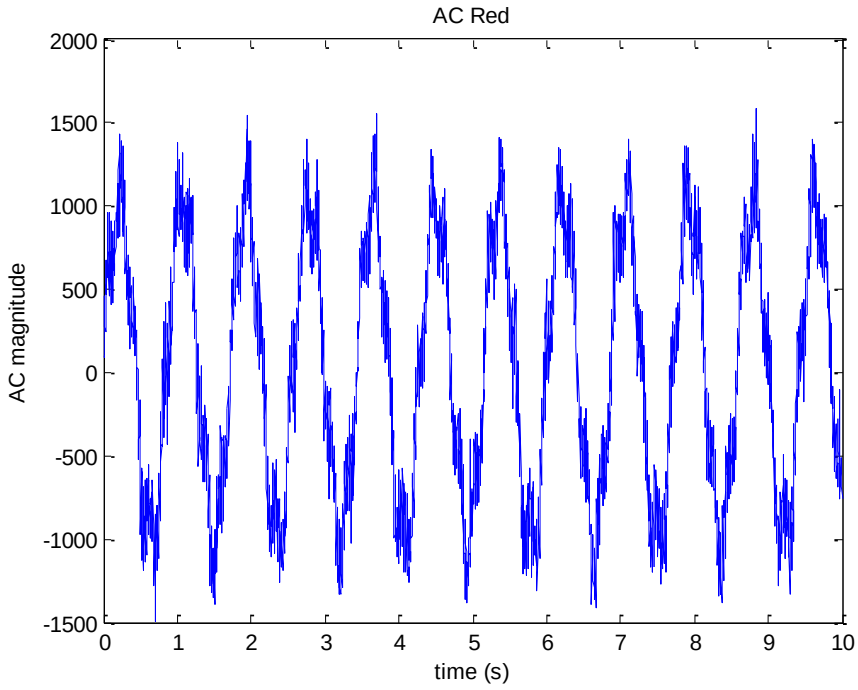


FIGURE 7 : EXEMPLE DE COMPOSANTE AC_R RECADREE AUTOUR DE 0

2.2 FILTRAGE DES SIGNAUX

2.2.1 FIR

Les composantes AC des mesures en rouge et infrarouge sont filtrées au moyen d'un filtre à réponse finie (FIR) [2]. Ce filtre a pour but d'éliminer les composantes hautes fréquences et ne garder que le signal utile qui est sensé se trouver autour d'une fréquence de 1 Hz.

On rappelle qu'un filtre FIR se représente sous la forme de sa transformée en Z de la façon suivante :

$$H(z) = \sum_{k=0}^{L-1} h_k \cdot z^{-k}$$

Où $(h_0, h_1, \cdots, h_{L-1})$ sont les coefficients du filtre et L l'ordre du filtre. Par définition la relation entre le signal de sortie y[n] et le signal d'entrée x[n] s'exprime comme suit :

$$y[n] = \sum_{k=0}^{L-1} h_k \cdot x[n-k]$$

Pour le projet, nous choisirons un filtre de type fenêtre de Hamming d'ordre 51 dont les coefficients sont donnés dans le Tableau 3.

k	h_k	k	h_k	k	h_k
0	1.4774946e-019	17	3.1294938e-002	34	2.7892178e-002
1	1.6465231e-004	18	3.4578348e-002	35	2.4459630e-002
2	3.8503956e-004	19	3.7651889e-002	36	2.1082435e-002
3	7.0848037e-004	20	4.0427695e-002	37	1.7838135e-002
4	1.1840522e-003	21	4.2824111e-002	38	1.4793934e-002
5	1.8598621e-003	22	4.4769071e-002	39	1.2004510e-002
6	2.7802151e-003	23	4.6203098e-002	40	9.5104679e-003
7	3.9828263e-003	24	4.7081811e-002	41	7.3374938e-003
8	5.4962252e-003	25	4.7377805e-002	42	5.4962252e-003
9	7.3374938e-003	26	4.7081811e-002	43	3.9828263e-003
10	9.5104679e-003	27	4.6203098e-002	44	2.7802151e-003
11	1.2004510e-002	28	4.4769071e-002	45	1.8598621e-003
12	1.4793934e-002	29	4.2824111e-002	46	1.1840522e-003
13	1.7838135e-002	30	4.0427695e-002	47	7.0848037e-004
14	2.1082435e-002	31	3.7651889e-002	48	3.8503956e-004
15	2.4459630e-002	32	3.4578348e-002	49	1.6465231e-004
16	2.7892178e-002	33	3.1294938e-002	50	1.4774946e-019

TABLEAU 3 : COEFFICIENTS DU FILTRE FIR

En langage C, les coefficients du filtre sont stockés sous la forme d'un tableau. Pour ne pas avoir à recopier les valeurs, vous trouverez sur l'intranet dans la section « Ressources », un code en langage C déclarant les coefficients du filtre.

On montre que ce filtre est de type passe- bas avec une fréquence de coupure de 10 Hz. Avec une fréquence d'échantillonnage de Fs = 500 Hz, on obtient les réponses en temps et en fréquence suivantes :

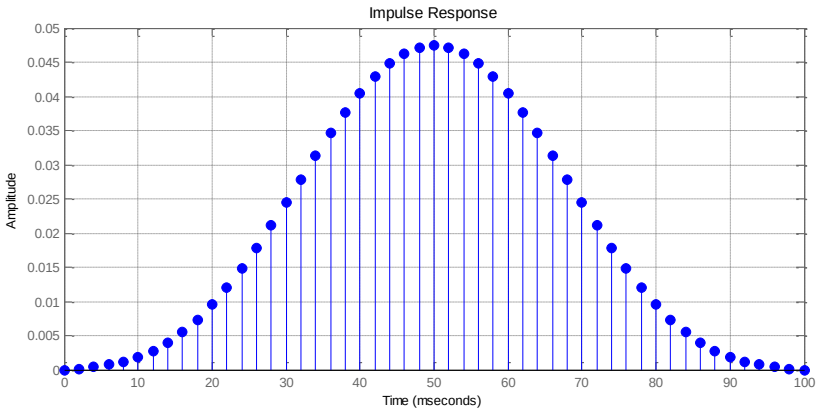


FIGURE 8: REPONSE IMPULSIONNELLE DU FILTRE FIR

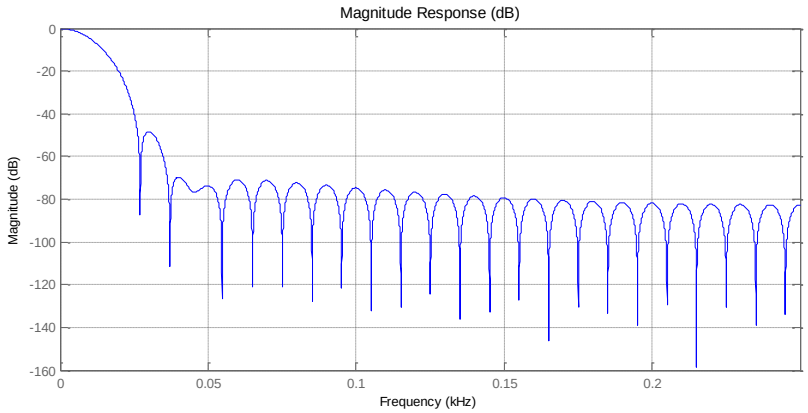


FIGURE 9 : REPONSE FREQUENTIELLE DU FILTRE FIR

On rappelle qu'il convient de filtrer seulement les composantes AC_R et AC_{IR} et de garder inchangés les composantes DC_R et DC_{IR}.

Pour réaliser ce type de filtre, il est d'usage d'utiliser un buffer permettant de mémoriser les L-1 derniers échantillons x(k) ainsi que l'échantillon courant x(n). Une fois la valeur y(n) calculée, le buffer est décalé vers la droite afin d'insérer le nouvel échantillon entrant comme le montre la Figure 10.

Afin d'éviter de recopier manuellement les échantillons lors de chaque décalage on pourra utiliser le principe du buffer circulaire (mais ce n'est pas une obligation !!!).

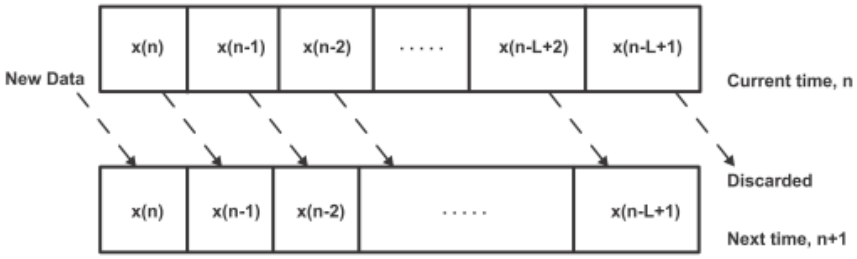


FIGURE 10 : BUFFER A DECALAGE

La Figure 11 montre un exemple de résultat de la composante AC_R obtenue après filtrage FIR.

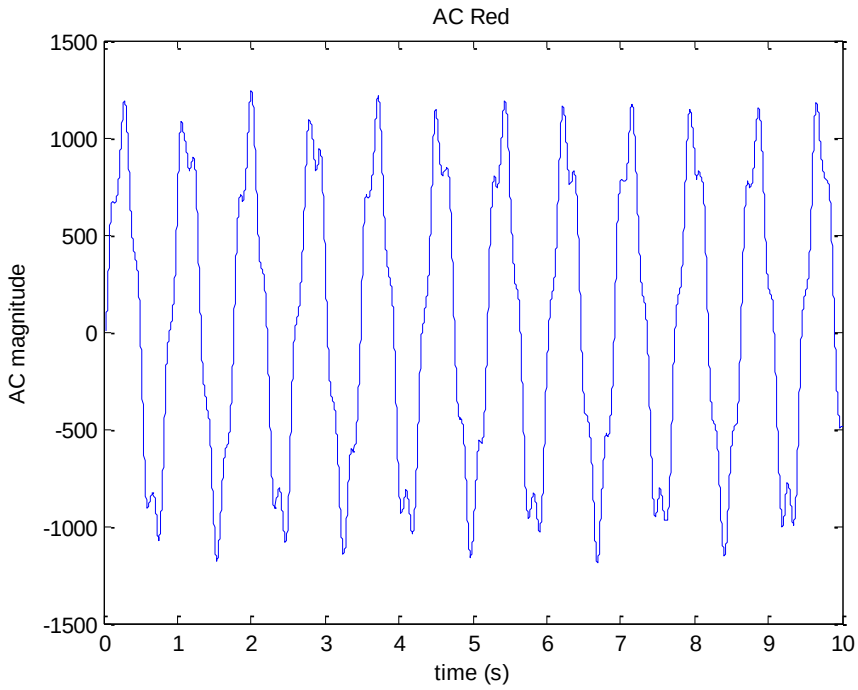


FIGURE 11 : EXEMPLE DE COMPOSANTE AC_R APRES FILTRAGE FIR

2.2.2 IIR

Après filtrage FIR, les composantes AC résultantes sont ensuite traitées au moyen d'un filtre à réponse infinie (IIR) [3]. Ce filtre sert à supprimer la composante continue résultante. Dans le projet nous utiliserons le filtre de transformée en Z suivante :

$$H(z) = \frac{1 - z^{-1}}{1 - \alpha \cdot z^{-1}}$$

On rappelle que la transformée en Z précédente implique la relation suivante entre le signal de sortie y[n] et le signal d'entrée x[n] :

$$y(n) = x[n] - x[n-1] + \alpha \cdot y[n-1]$$

Dans le projet nous choisirons une valeur de α égale à 0.992

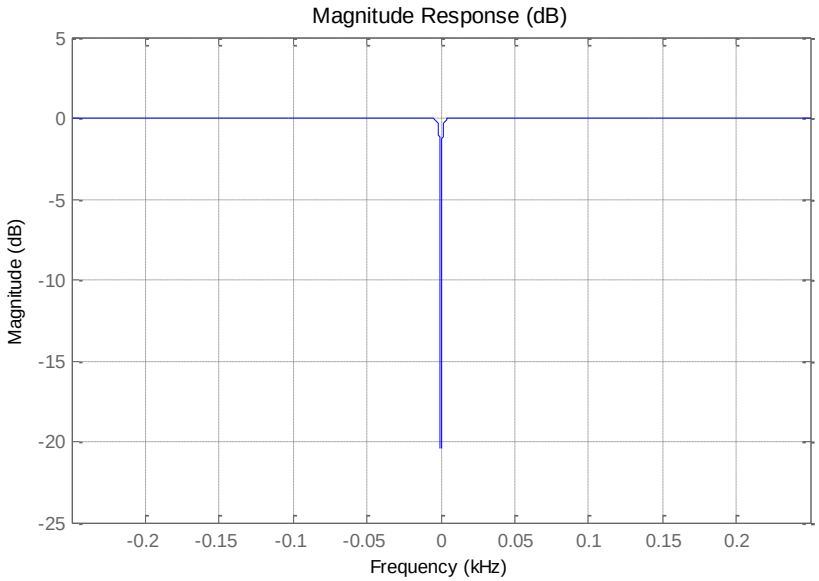


FIGURE 12 : REPONSE FREQUENTIELLE DU FILTRE IIR

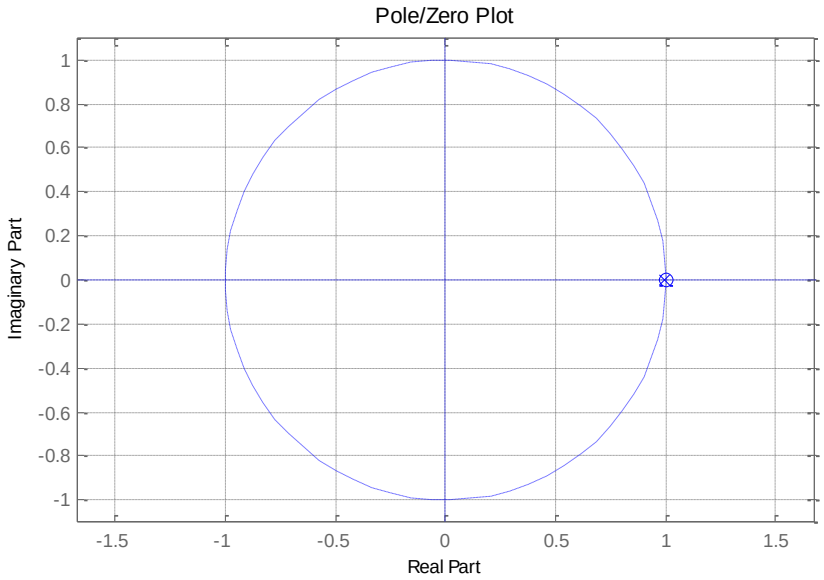


FIGURE 13 : EMPLACEMENT DES POLES ET DES ZEROS DU FILTRE IIR

La Figure 14 montre un exemple de résultat obtenu sur la composante AC_R après filtrage IIR.

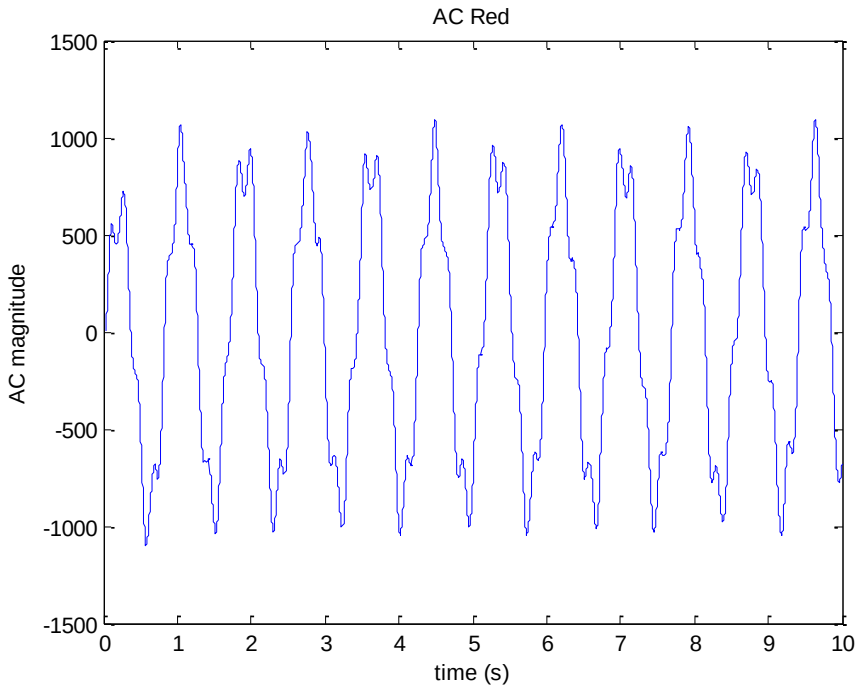


FIGURE 14 : EXEMPLE DE COMPOSANTE AC_R APRES FILTRAGE IIR

On rappelle qu'il convient de filtrer seulement les composantes AC_R et AC_{IR} et de garder inchangés les composantes DC_R et DC_{IR}.

2.3 MESURES

2.3.1 Calcul de SP02

Le taux de saturation de l'oxygène dans le sang s'obtient à partir des valeurs filtrées de AC_R et AC_{IR} ainsi que des valeurs DC_R et DC_{IR} en calculant la rapport suivant :

$$RsIR = \frac{\frac{\mathrm{PtP}(AC_R)}{DC_R}}{\frac{\mathrm{PtP}(AC_{IR})}{DC_{IR}}}$$

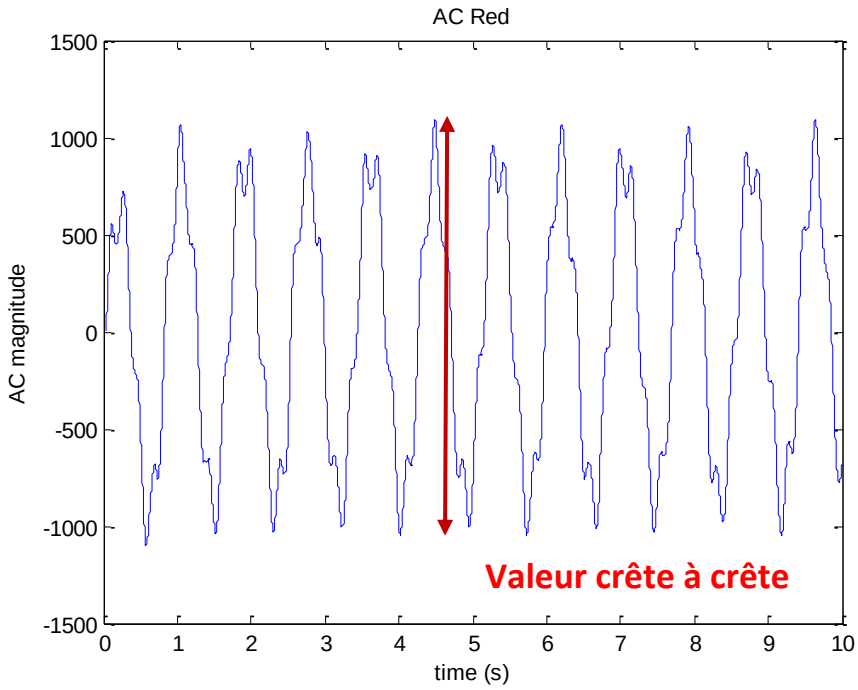


FIGURE 15 : CALCUL DE LA VALEUR CRETE A CRETE DE LA COMPOSANTE AC R

Où $\mathrm{PtP}(x)$ correspond à la valeur crête à crête (peak-to-peak en anglais) du signal x.

On utilise ensuite la table de correspondance suivante :

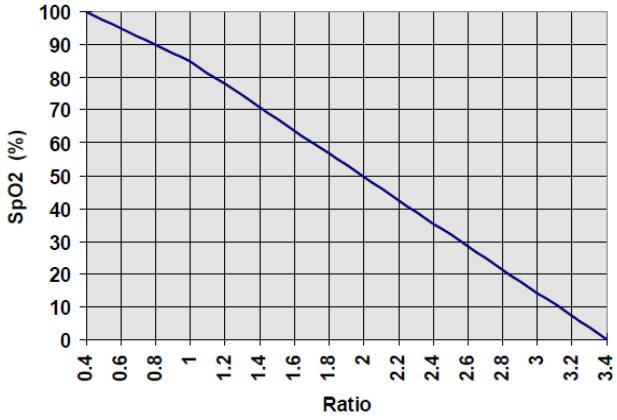


FIGURE 16 : CORRESPONDANCE ENTRE RSIR ET SPO2

2.3.2 Calcul du rythme cardiaque

Le pouls se mesure en estimant la fréquence des signaux AC_R et AC_{IR} (qui doit être à peu de chose près identique).

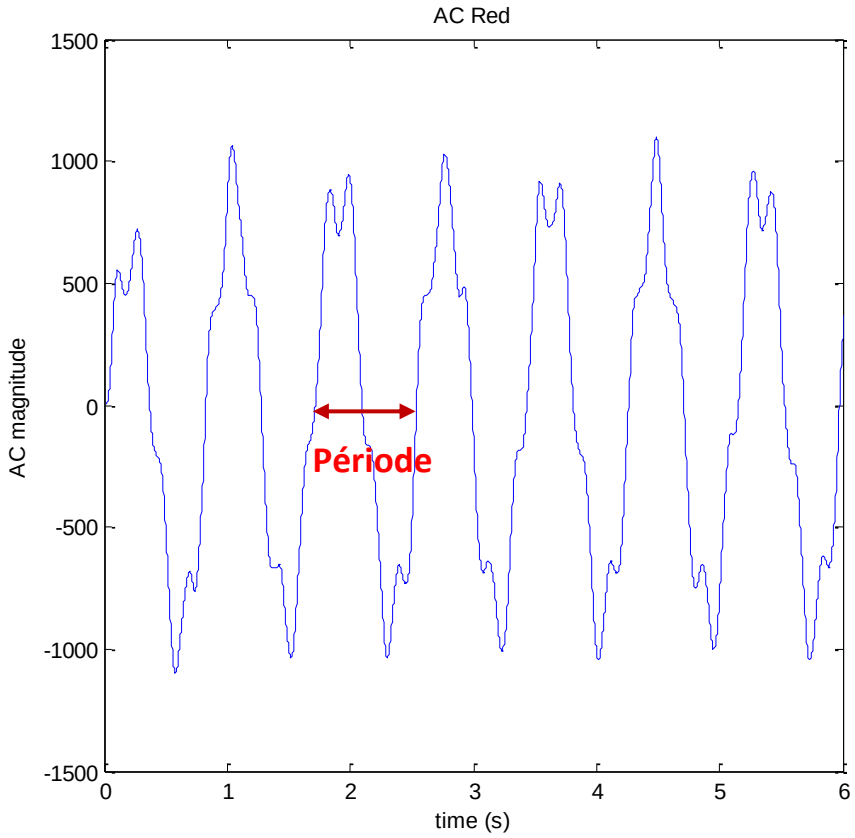


FIGURE 17 : CALCUL DU RYTHME CARDIAQUE SUR LA COMPOSANTE AC_R

2.4 AFFICHAGE

Une interface utilisateur a été développée en langage Python à votre intention :

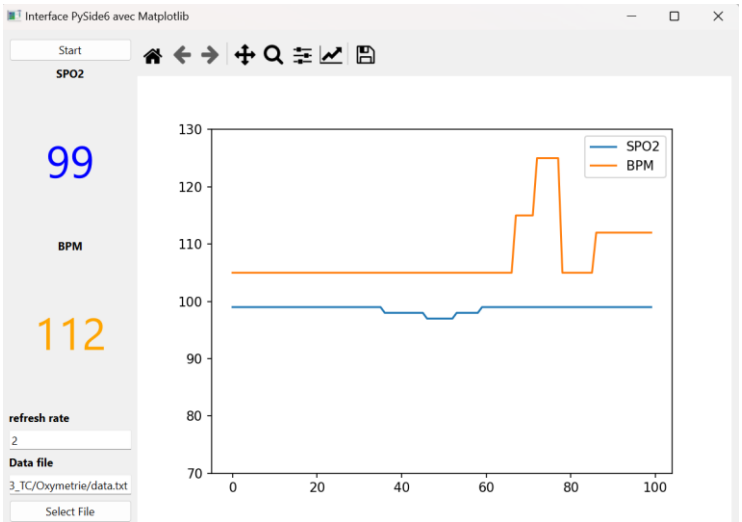


FIGURE 18 : INTERFACE UTILISATEUR FOURNIE

Les données qui apparaissent sur cette interface sont lues dans le fichier `data.txt` (mise à jour toutes les secondes) :

- taux de saturation de l'oxygène dans le sang (SPO2) : type entier
- rythme cardiaque (BPM) : type entier

Afin d'être sûr de ne pas avoir de problèmes de concurrence entre les programmes, les accès en lecture ou écriture au fichier Data.txt sont protégés par un verrou. Il n'est pas possible de lire et écrire en même temps dans un fichier. C'est ce qu'on appelle l'exclusion mutuelle. Ce mécanisme de verrou est mis en œuvre le fichier .verrouData La présence du fichier verrou indique qu'une opération de lecture ou écriture est en cours sur le fichier de données correspondant. L'absence de verrou autorise la lecture ou l'écriture. Ne pas oublier la gestion du verrou !!!

3 COTE PRATIQUE

3.1 ENVIRONNEMENT DE TRAVAIL

3.1.1 Outil de compilation

Pour la partie simulation, l'environnement de travail n'est pas imposé, veuillez-vous reporter aux consignes données dans le module « Algorithme et Langage C ».

Pour la partie connexion à la carte électronique, les drivers USB sont prévus pour fonctionner sous Windows uniquement. Un développement sous windows devient ainsi nécessaire. Nous vous conseillons d'installer MSYS2 ainsi que le compilateur MinGW64 (https://www.msys2.org/). A partir de MSYS2 les installations de gcc et make se font de la façon suivante :

```
pacman -S mingw-w64-ucrt-x86_64-gcc
pacman -S make
```

(plus de détail sur le tuto d'installation sur Moodle)

Notez qu'il est possible d'utiliser la version de MinGW inclue dans Clion par exemple, mais cela vous empêchera d'installer simplement de nouveau package comme le permet MSYS2.

Afin de faciliter la recette on vous demande d'organiser les fichiers de votre projet de façon à avoir une architecture identique à la Figure 19. Attention toute architecture différente pourra entrainer un malus lors de la recette

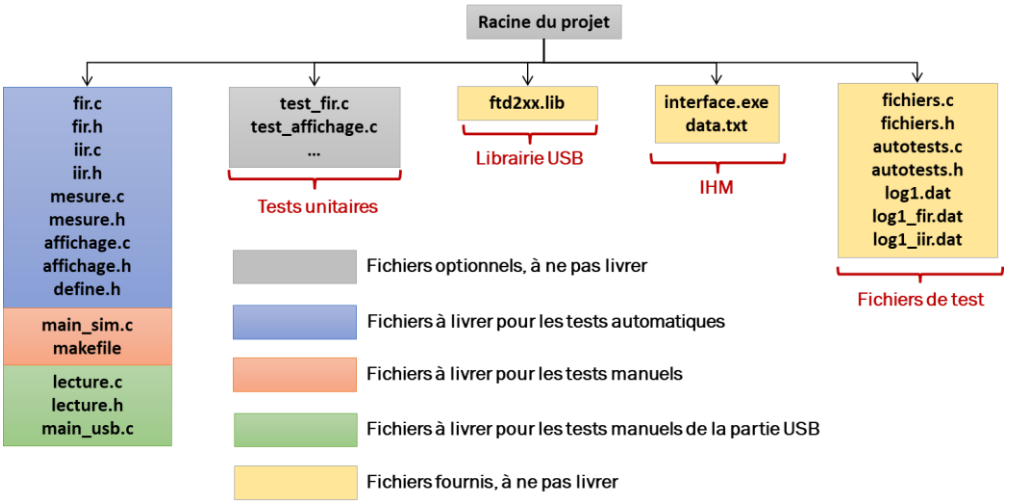


FIGURE 19 : ARBORESCENCE DE TRAVAIL

3.1.2 IHM d'affichage

3.1.3 Installation

Sous Windows (conseillé), téléchargez le fichier exécutable interface.exe. Lancez-le via un double clic. Si Windows vous demande si le fichier est sûr, acceptez. Vous n'avez pas besoin d'IDE (pycharm, VS Code) ni de bibliothèque Python particulière (MatPlotLib, PySide 6) pour cette installation qui se veut simple et autonome.

Avec un OS MAC ou Linux, vous ne pouvez pas utiliser interface.exe. Il faudra télécharger le fichier python interface.py, puis lancer le fichier via python, soit avec un IDE (pycharm, VSCode) déjà configuré soit en ligne de commande. La bibliothèque PySide 6 ainsi que MatplotLib sont requises et doivent être installées via l'outil `pip` en ligne de commande ou via votre IDE.

Une seule installation de cette IDE peut suffire pour un binôme si des difficultés persistent.

3.1.4 Utilisation de l'interface

Une fois lancée l'interface propose à droite un graphique, à gauche deux valeurs mises à jour périodiquement. En haut à gauche se situe un bouton start/stop : il permet de lire ou d'arrêter de lire le fichier « `data.txt` » toutes les 2,5 secondes environ. Il n'est pas possible de modifier la période de lecture. L'interface ne lira pas le fichier « `data.txt` » si le fichier « `.verrouData` » est présent sur votre disque. Pensez donc à le supprimer à la main le cas échéant avant vos tests. Normalement, vous pouvez laisser l'interface lancée et n'utiliser que le bouton start/stop pour afficher les données. Cette interface n'est qu'une

aide graphique, elle ne substitue en aucun cas aux tests que vous devez effectuer pendant le projet.

3.1.5 Carte électronique

L'utilisation de la carte électronique est réservée au binôme qui vont transférer les données par USB. Il n'y a que quelques cartes par groupe. Demandez à votre professeur référent de vous fournir le matériel.

Branchements :

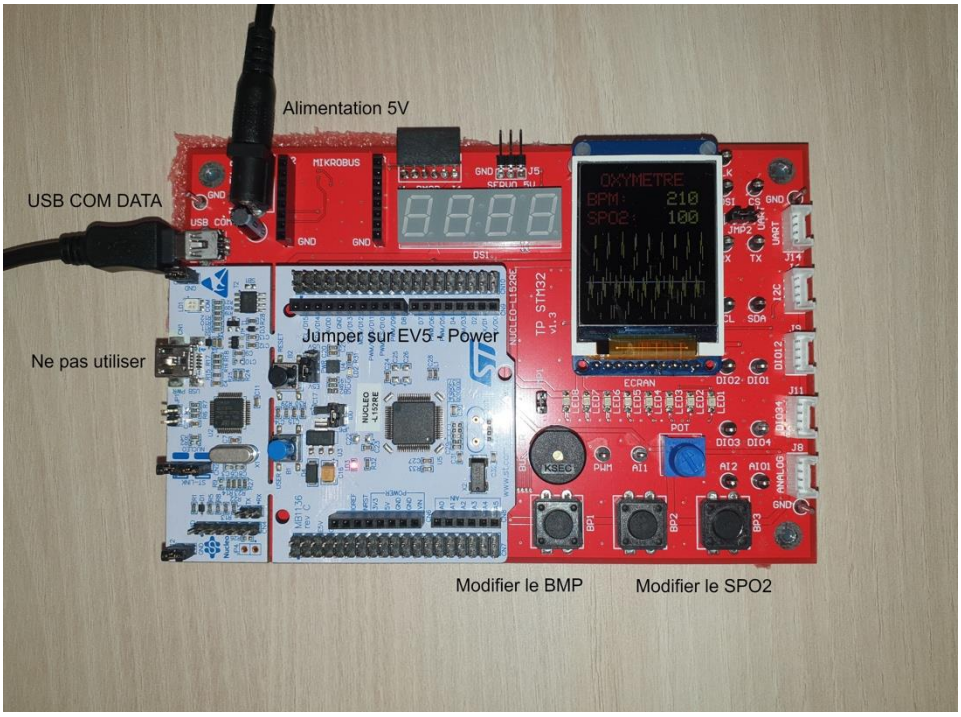
Comme l'indique la figure suivante, l'alimentation s'effectue avec un câble jack 5V fourni par votre professeur. Ne pas alimenter la carte d'une autre facon. En cas de doute, demander à votre professeur.

Les datas circulent via USB par le port USB COM. Vous brancherez la partie USB sur votre PC sur un port libre.

Ne pas utiliser le port micro USB de la carte STM32

Le jumper de la carte STM32 doit être mis sur EV5 PWR pour utiliser l'alimentation fournie pour le projet. Si ce n'est pas le cas, contactez votre professeur.

Vous pouvez modifier le BPM et le SPO2 en appuyant sur les boutons, les valeurs cyclent sur des constantes prédéfinies. Utilisez-les pour tester vos filtres !



3.2 EVALUATION

3.2.1 Rendu

La recette de votre programme sera effectuée le dernier jour entre 14:00 et 18:00. Les fichiers à inclure sont détaillés en Figure 20.

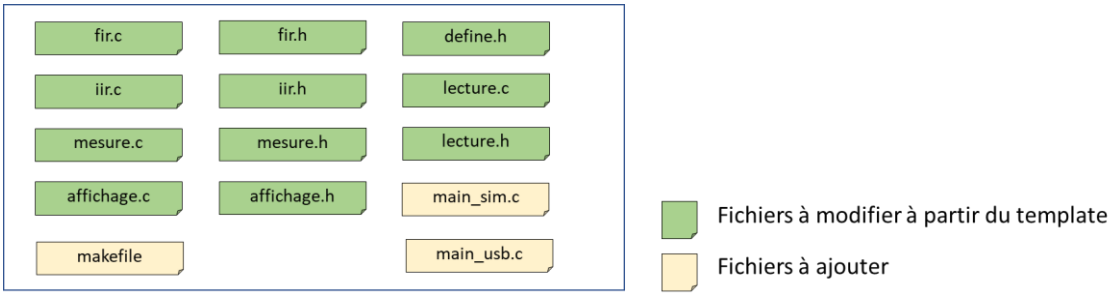


FIGURE 20 : SOURCE A RENDRE

3.2.2 Evaluation par compétences

Les compétences évaluées dans le projet ainsi que les méthodes évaluations associées sont décrites dans le Tableau 4.

Méthode d'évaluation	Compétences évaluées	Principe de validation	Pondération
Recette	L'étudiant·e sait développer des fonctions	tests automatiques Blocs FIR, IIR, Affichage et calcul	4,5

	en respectant un cahier des charges		
Recette	L'étudiant·e sait intégrer différentes fonctions	Tests manuels en simulation et USB*	3
QCM papier	L'étudiant·e maitrise son sujet sur les aspects techniques.	Note de QCM	2
QCM papier	L'étudiant·e maitrise le langage de programmation	Note de QCM	2
QCM papier	L'étudiant·e est capable de prendre du recul sur le sujet du projet	Note de QCM	2
Audit	L'étudiant·e est capable d'expliquer l'entrée/la sortie (le type et le sens) d'une fonction	Questions aux étudiants	1,5
Audit	L'étudiant·e est capable d'expliquer, de manière globale, le fonctionnement global d'une fonction (son déroulé, son utilité globale)	Questions aux étudiants	1,5
Audit	L'étudiant·e est capable d'expliquer une fonction ligne par ligne	Questions aux étudiants	1,5
Audit	L'étudiant·e se soucie de la suite de son projet (documentation…)	Audit de code	1
Audit	L'étudiant·e sait respecter des consignes	Audit de fichiers (Noms des fonctions, nommage des fichiers, makefile, …)	1
Total			**20**

TABLEAU 4 : COMPETENCES EVALUEES

*Pour les cycles autres que CIR3, la partie USB est optionnelle.

Le tableau croisé faisant le lien entre les compétences évaluées et les compétences ISEN Ouest est donné dans le Tableau 5.

	RNCP13040BC01 Définir les exigences et l'architecture du système électronique ou informatique			RNCP13040BC02 Réaliser le système électronique ou informatique			RNCP13040BC03 Piloter des hommes et des projets de systèmes électronique ou informatique					
	LC1	LC2	LC3	LC1	LC2	LC3	LC1	LC2	LC3	LC4	LC5	LC6
	Etablir l'expression du besoin client	Piloter le développement	Prendre en compte toutes les contraintes	Modéliser le système	Développer la partie technique	Anticiper et se former	Identifier les objectifs	Gérer le projet	Déployer les méthodes	Gérer l'atteinte des objectifs	Gérer la clôture	Réaliser retour d'expérience
L'étudiant·e sait developper des fonctions en respectant un cahier des charges		x	x		x		x	x	x	x		
L'étudiant·e sait intégrer différentes fonctions		x			x			x	x	x		
L'étudiant·e maitrise son sujet sur les aspects techniques.				x		x						
L'étudiant·e maitrise le langage de programmation					x	x						
L'étudiant·e est capable de prendre du recul sur le sujet du projet			x	x		x						
L'étudiant·e est capable d'expliquer l'entrée/la sortie (le type et le sens) d'une fonction				x	x						x	x
L'étudiant·e est capable d'expliquer, de manière globale, le fonctionnement global d'une fonction (son déroulé, son utilité globale)				x	x						x	x
L'étudiant·e est capable d'expliquer une fonction ligne par ligne				x	x						x	x
L'étudiant·e se soucie de la suite de son projet (documentation...)									x		x	x
L'étudiant·e sait respecter des consignes									x		x	x

TABLEAU 5 : TABLEAU CROISE DES COMPETENCES

3.2.3 Tests automatiques

Pour tester automatiquement les blocs du projet, nous mettons à disposition un fichier de fonctions (autotests.c/autotests.h) permettant de tester unitairement les différents blocs :

- testAffichage()
- testFIR()
- testIIR()
- testMesure()

Chaque fonction affiche une valeur entre 1 (bloc fonctionnel) et 0 (bloc non fonctionnel). Un programme permettant de tester unitairement les différents blocs est le suivant :

3.3 FONCTIONS IMPOSEES POUR LES TESTS AUTOMATIQUES

Une archive contenant tous les fichiers demandés et les prototypes associés vous est fournie sur l'intranet sous le nom Minimal.zip.

Nous vous conseillons de partir de ces fichiers et de compléter au fur et à mesure les différentes fonctions.

Si vous n'avez pas réussi un bloc, commentez le code qui n'est pas fonctionnel mais laissez le squelette obligatoire de la fonction (coquille vide).

3.3.1 Fichier de définition « define.h »

```
#ifndef DEFINE_H
#define DEFINE_H

typedef struct{
          float acr;   /*!< AC R */
          float dcr;   /*!< DC R */
          float acir;   /*!< AC IR */
          float dcir;   /*!< DC IR */
} absorp;
typedef struct{
          int spo2;   /*!< SPO2 */
          int pouls;   /*!< Pouls */
} oxy;

#endif
```

3.3.2 FIR

Le prototype de la fonction `FIR` n'est pas imposé, ceci dans le but de laisser au libre choix des étudiants les paramètres d'entrées et de sorties du bloc. En revanche, il est demandé de fournir une fonction de test de ce bloc FIR nommée « `firTest` » dont le prototype est imposé :

fir.h

```
absorp firTest(char* str);
```

fir.c

```
absorp firTest(char* str){
  absorp data;
  …
  return data;
}
```

Cette fonction ouvre et lit les valeurs d'absorption contenues dans le fichier nommé `str` et filtre ces valeurs au moyen de la fonction `FIR`. Lorsque le fichier est fini d'être lu, la fonction retourne la dernière valeur d'absorption filtrée. Attention pour avoir le maximum de points il faut que la fonction `firTest` appelle itérativement la fonction `FIR` et que cette fonction contienne l'algorithme de filtrage FIR.

3.3.3 IIR

Le prototype de la fonction `IIR` n'est pas imposé, ceci dans le but de laisser au libre choix des étudiants les paramètres d'entrées et de sorties du bloc. En revanche, il est demandé de fournir une fonction de test de ce bloc IIR nommée « `iirTest` » dont le prototype est imposé :

iir.h

```
absorp iirTest(char* str);
```

iir.c

```
absorp iirTest(char* str){
  absorp data;
  …
  return data;
}
```

Cette fonction ouvre et lit les valeurs d'absorption contenues dans le fichier nommé `str` et filtre ces valeurs au moyen de la fonction `IIR`. Lorsque le fichier est fini d'être lu, la fonction retourne la dernière valeur d'absorption filtrée. Attention pour avoir le maximum de points il faut que la fonction `iirTest` appelle itérativement la fonction `IIR` et que cette fonction contienne l'algorithme de filtrage IIR.

3.3.4 Mesure

Le prototype de la fonction `mesure` n'est pas imposé, ceci dans le but de laisser au libre choix des étudiants les paramètres d'entrées et de sorties du bloc. En revanche, il est demandé de fournir une fonction de test de ce bloc Mesure nommée « `mesureTest` » dont le prototype est imposé :

mesure.h

```
oxy mesureTest(char* str);
```

mesure.c

```
oxy mesureTest(char* str){
  oxy myOxy;
  …
  return myOxy;

}
```

Cette fonction ouvre et lit les valeurs d'absorption contenues dans le fichier nommé `str` et calcules des valeurs d'oxymétrie au moyen de la fonction `mesure`. Lorsque le fichier est fini d'être lu, la fonction retourne la dernière valeur d'oxymétrie calculée. Attention pour avoir le maximum de points il faut que la fonction `mesureTest` appelle itérativement la fonction `mesure` et que cette fonction contienne l'algorithme de calcul d'oxymétrie et de fréquence cardiaque.

3.3.5 Affichage

affichage.h

```
void affichage(oxy myOxy);
```

affichage.c

```
void affichage(oxy myOxy){
  …
}
```

Si le verrou est présent, la fonction ne modifie pas le fichier `data.txt`.

3.4 FICHIERS DE TESTS

Des fichiers de test vous sont fournis afin de vous donner la possibilité de tester vos blocs de façon unitaire :

- log1.dat : en sortie de fonction lecture et après recentrage
- log1_fir.dat : en sortie de filtrage FIR
- log1_iir.dat : en sortie de filtrage IIR

La lecture des fichiers est possible en utilisant les fonctions disponibles dans fichiers.h/fichiers.c, dont voici un exemple d'utilisation :

```
absorp myAbsorp;
int fileState = 0;
FILE* pf=initFichier("record1.dat");
while(fileState != EOF){
      myAbsorp = lireFichier(pf,&fileState);
}
finFichier(pf);
```

Pour le fichier, log1_iir.dat, les valeurs calculées de SpO2 et de pouls sont les suivantes :

- Sp02 = 98 %
- Pouls = 80 bpm

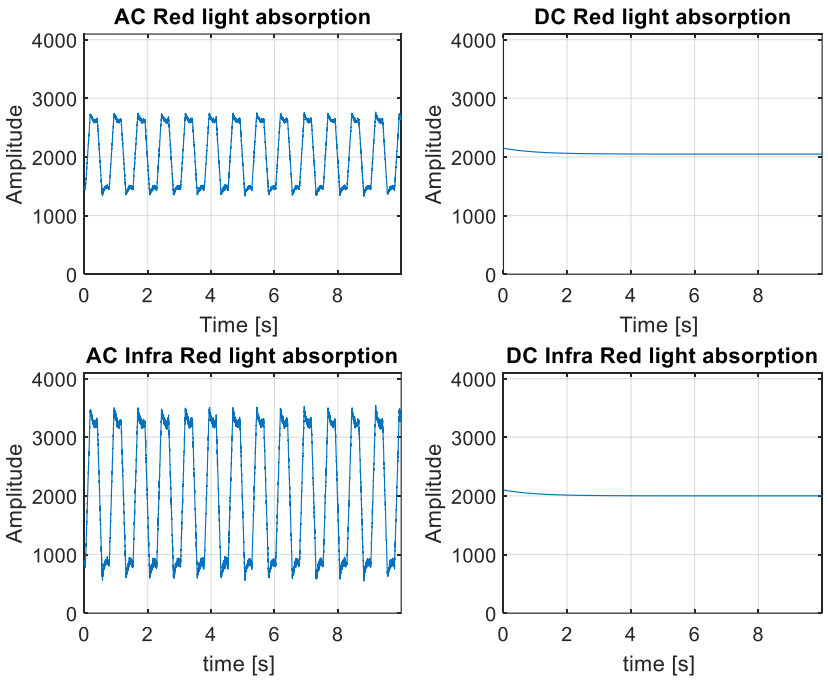


FIGURE 21 : APPERCU DU FICHIER LOG1.DAT

4 REFERENCES

[1] J. Corbel, « Présentation générale du projet A3 : oxymètre de Pouls », ISEN Ouest, Février 2020.

[2] « Finite impulse Response filter », Wikipedia, the free encyclopedia

[3] « Infinite Impulse Response filter », Wikipedia, the free encyclopedia

[4] Driver FTDI chip, Future Technology Devices International Ltd.